In [1]:
```python
# Imports
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
import numpy as np
import pandas as pd
from wordcloud import WordCloud
from ucimlrepo import fetch_ucirepo

pd.options.mode.chained_assignment = None
```

In [2]:
```python
# fetch dataset from the UCI archive website by its ID.
drug_dataset_id = 462
Drugs = fetch_ucirepo(id=drug_dataset_id)
```

In [3]:
```python
# variable info.
print(Drugs.metadata["additional_info"]["variable_info"])
```

```
1. drugName (categorical): name of drug
2. condition (categorical): name of condition
3. review (text): patient review
4. rating (numerical): 10 star patient rating
5. date (date): date of review entry
6. usefulCount (numerical): number of users who found review useful
```

In [4]:
```python
# Create a total data from the Drugs
data_original = Drugs.data.original

# Checking for missing values in the dataset
missing_values = data_original.isnull().sum()

# Drop the rows with the missing condition value.
data_original = data_original.dropna()

# Basic summary statistics for numerical columns
numerical_summary = data_original.describe()

# Distribution of unique values for categorical columns
drug_count = data_original['drugName'].nunique()
condition_count = data_original['condition'].nunique()

# Summarize text review column (average length of reviews)
data_original['review_length'] = data_original['review'].apply(lambda x: ler

average_review_length = data_original['review_length'].mean()

# Display key stats
print(f"Missing Values \n{missing_values}\n")
print(f"Numerical Summary: \n{numerical_summary}\n")
print(f"Number of unique drugs: {drug_count}")
print(f"Number of unique conditions: {condition_count}")
print(f"Average length of review: {round(average_review_length)}")
```

```
Missing Values
id                0
drugName          0
condition      1194
review            0
rating            0
date              0
usefulCount       0
dtype: int64

Numerical Summary:
                  id         rating    usefulCount
count  213869.000000  213869.000000  213869.000000
mean   116076.924786       6.991149      28.094118
std     67016.705794       3.275792      36.401377
min         0.000000       1.000000       0.000000
25%     58122.000000       5.000000       6.000000
50%    115972.000000       8.000000      16.000000
75%    174018.000000      10.000000      36.000000
max    232291.000000      10.000000    1291.000000

Number of unique drugs: 3667
Number of unique conditions: 916
Average length of review: 85
```

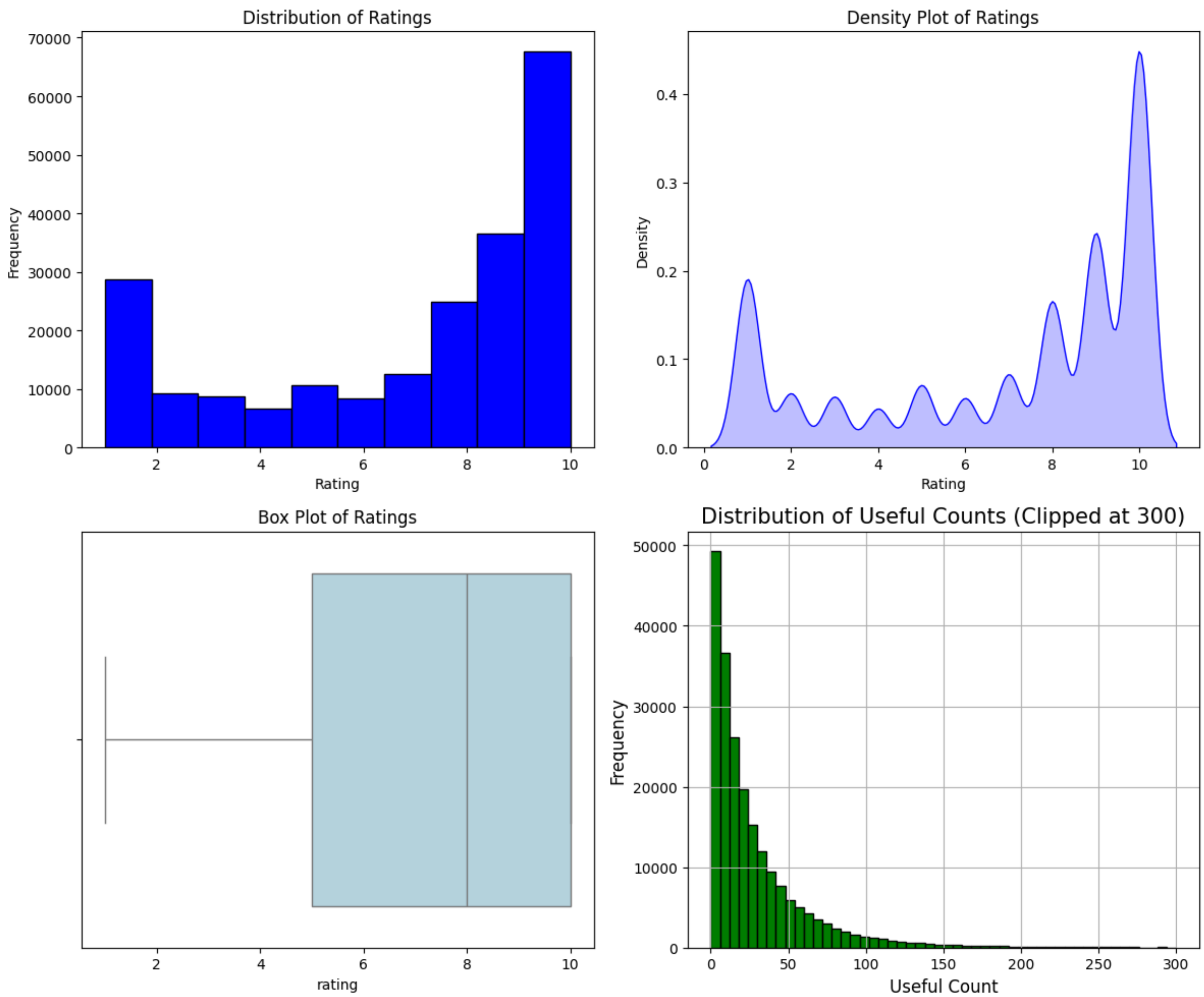
In [5]:

```python
# Convert 'date' column to datetime format
data_original['date'] = pd.to_datetime(data_original['date'], format="%d-%b-

# Extract year from the date
data_original['year'] = data_original['date'].dt.year
# Extract the month from the date
data_original['month'] = data_original['date'].dt.month

# Group data by year and month, and calculate the number of reviews and aver
data_original['year_month'] = data_original['date'].dt.to_period('M')
```

In [6]:

```python
# Histogram for numerical variables (rating and usefulCount, assuming they e
fig, axs = plt.subplots(2, 2, figsize=(12, 10))

axs[0, 0].hist(data_original["rating"], bins=10, edgecolor="black", color="b
axs[0, 0].set_title('Distribution of Ratings')
axs[0, 0].set_xlabel('Rating')
axs[0, 0].set_ylabel('Frequency')

sns.kdeplot(data_original['rating'], fill=True, color='blue', ax=axs[0, 1])
axs[0, 1].set_title('Density Plot of Ratings')
axs[0, 1].set_xlabel('Rating')
axs[0, 1].set_ylabel('Density')
plt.grid(True)

# Box plots to show spread and outliers
sns.boxplot(x=data_original["rating"], color="lightblue", ax=axs[1, 0])
axs[1, 0].set_title("Box Plot of Ratings")

# Distribution of Useful counts plot
axs[1, 1].hist(data_original['usefulCount'], bins=50, color='green', edgecol
```

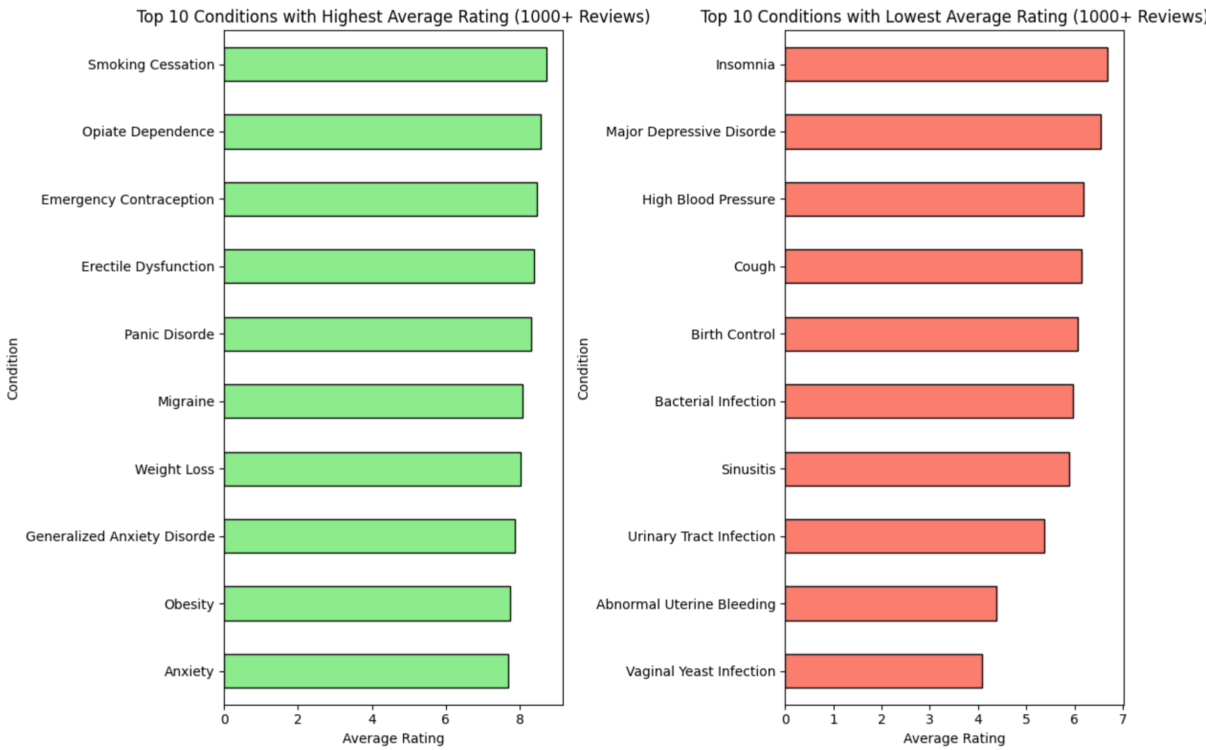
```
axs[1, 1].set_title('Distribution of Useful Counts (Clipped at 300)', fontsi
axs[1, 1].set_xlabel('Useful Count', fontsize=12)
axs[1, 1].set_ylabel('Frequency', fontsize=12)
axs[1, 1].grid(True)

plt.tight_layout()
plt.show()
```



In [7]:

```
# Calculate the average rating for each condition
avg_rating_per_condition = data_original.groupby("condition")["rating"].mear
condition_review_counts = data_original.groupby("condition").size()  # Count

conditions_at_least_1000_reviews = condition_review_counts[condition_review_
filtered_avg_ratings = avg_rating_per_condition.loc[conditions_at_least_1000

# Sort the conditions by average rating
sorted_ratings = filtered_avg_ratings.sort_values()

# Visualize the top 10 highest and lowest rated conditions with at least 100
plt.figure(figsize=(12, 8))

# Plot the top 10 highest rated conditions
plt.subplot(1, 2, 1)
sorted_ratings.tail(10).plot(kind="barh", color="lightgreen", edgecolor="bla
plt.title("Top 10 Conditions with Highest Average Rating (1000+ Reviews)")
```

```python
plt.xlabel("Average Rating")
plt.ylabel("Condition")

# Plot the top 10 lowest rated conditions
plt.subplot(1, 2, 2)
sorted_ratings.head(10).plot(kind="barh", color="salmon", edgecolor="black")
plt.title("Top 10 Conditions with Lowest Average Rating (1000+ Reviews)")
plt.xlabel("Average Rating")
plt.ylabel("Condition")

# Display the plots
plt.tight_layout()
plt.show()
```

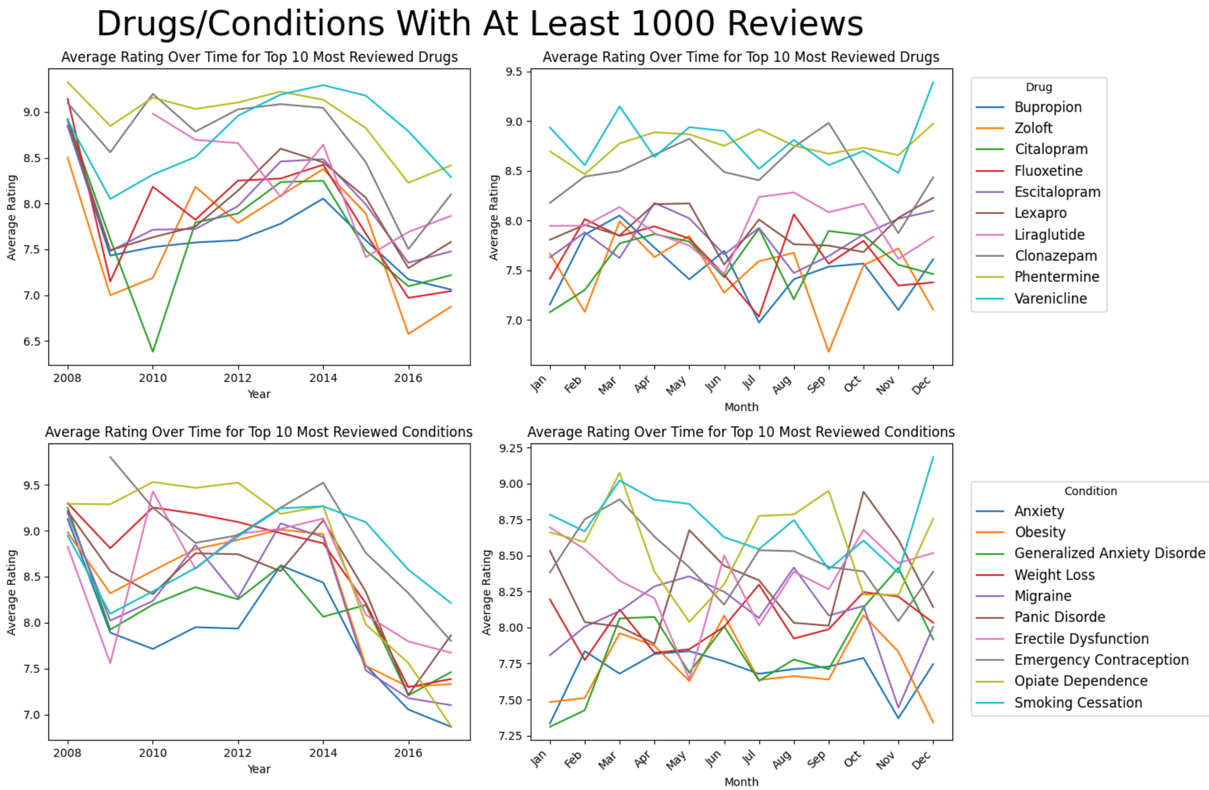


In [8]:

```python
# string arr of all of the months in a year, used for the x ticks
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oc
#Including the effect of time aka including the date information
avg_rating_per_drug = data_original.groupby("drugName")["rating"].mean()
drug_review_counts = data_original.groupby("drugName").size()

drugs_at_least_1000_reviews = drug_review_counts[drug_review_counts >= 1000]
filtered_avg_drug_ratings = avg_rating_per_drug.loc[drugs_at_least_1000_revi

# Sort the conditions by average rating
sorted_drug_ratings = filtered_avg_drug_ratings.sort_values()

# Identify the top 10 most used drugs (based on number of reviews)
top_10_drugs = sorted_drug_ratings.tail(10).index

top_10_conditions = sorted_ratings.tail(10).index

# Group by drug and year/month, then calculate the average rating
ratings_by_year_top_drugs = data_original[data_original['drugName'].isin(top
```

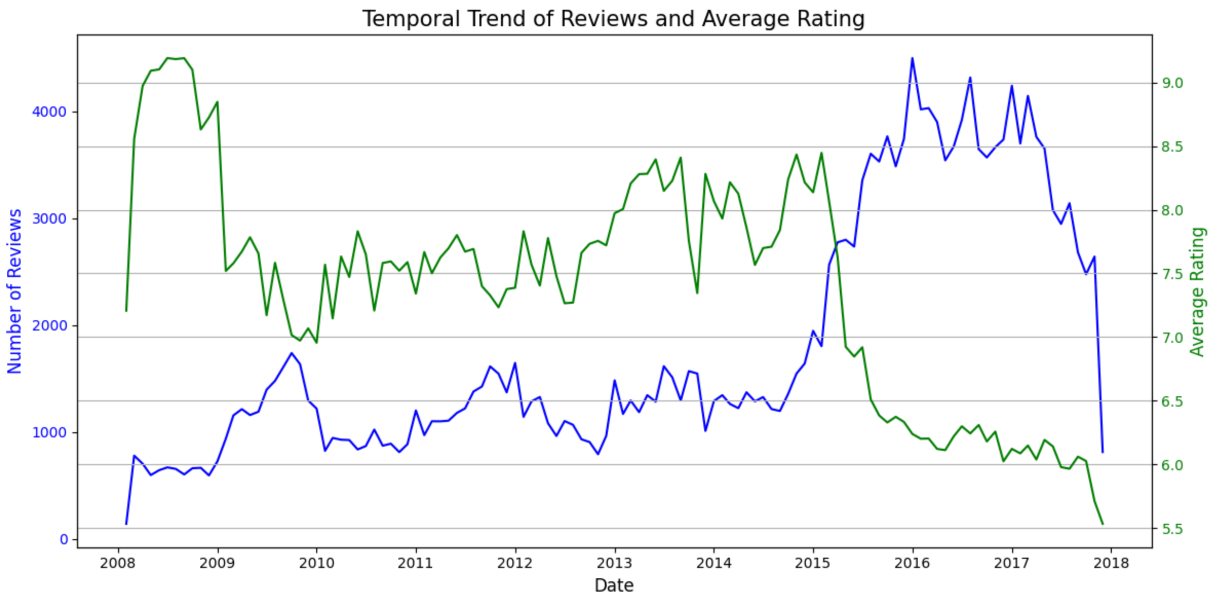
```
ratings_by_month_top_drugs = data_original[data_original['drugName'].isin(to

# Group by condition and year/month, then calculate the average rating
ratings_by_year_top_conditions = data_original[data_original['condition'].is
ratings_by_month_top_conditions = data_original[data_original['condition'].i


# Create a 2x2 grid of subplots
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
fig.suptitle('Drugs/Conditions With At Least 1000 Reviews', fontsize=30)

# Plotting for each drug and condition
for i in range(len(top_10_drugs)):
    drug = top_10_drugs[i]
    sns.lineplot(x='year', y='rating', data=ratings_by_year_top_drugs[rating
    sns.lineplot(x='month', y='rating', data=ratings_by_month_top_drugs[rati

    condition = top_10_conditions[i]
    sns.lineplot(x='year', y='rating', data=ratings_by_year_top_conditions[r
    sns.lineplot(x='month', y='rating', data=ratings_by_month_top_conditions

axs[0, 0].set_title('Average Rating Over Time for Top 10 Most Reviewed Drugs
axs[0, 0].set_xlabel('Year')
axs[0, 0].set_ylabel('Average Rating')

axs[0, 1].set_title('Average Rating Over Time for Top 10 Most Reviewed Drugs
axs[0, 1].set_xlabel('Month')
axs[0, 1].set_ylabel('Average Rating')
axs[0, 1].set_xticks(range(1, 13))  # Set ticks for each month (1 to 12)
axs[0, 1].set_xticklabels(months)

axs[1, 0].set_title('Average Rating Over Time for Top 10 Most Reviewed Condi
axs[1, 0].set_xlabel('Year')
axs[1, 0].set_ylabel('Average Rating')

axs[1, 1].set_title('Average Rating Over Time for Top 10 Most Reviewed Condi
axs[1, 1].set_xlabel('Month')
axs[1, 1].set_ylabel('Average Rating')
axs[1, 1].set_xticks(range(1, 13))  # Set ticks for each month (1 to 12)
axs[1, 1].set_xticklabels(months)

# Collect handles and labels from one of the subplots
handles_drugs, labels_drugs = axs[0, 0].get_legend_handles_labels()
handles_condition, labels_condition = axs[1, 0].get_legend_handles_labels()

# Remove the legends as it will be displayed on the side
axs[0, 0].legend().remove()
axs[0, 1].legend().remove()
axs[1, 0].legend().remove()
axs[1, 1].legend().remove()

# Set a common legend outside the plot (to the right)
fig.legend(handles_drugs, labels_drugs, loc='center left', bbox_to_anchor=(1
fig.legend(handles_condition, labels_condition, loc='center left', bbox_to_a

# Rotate the x-ticks to make them readable
```

```python
plt.setp(axs[0, 1].get_xticklabels(), rotation=45, ha="right")
plt.setp(axs[1, 1].get_xticklabels(), rotation=45, ha="right")

plt.tight_layout()
plt.show()
```

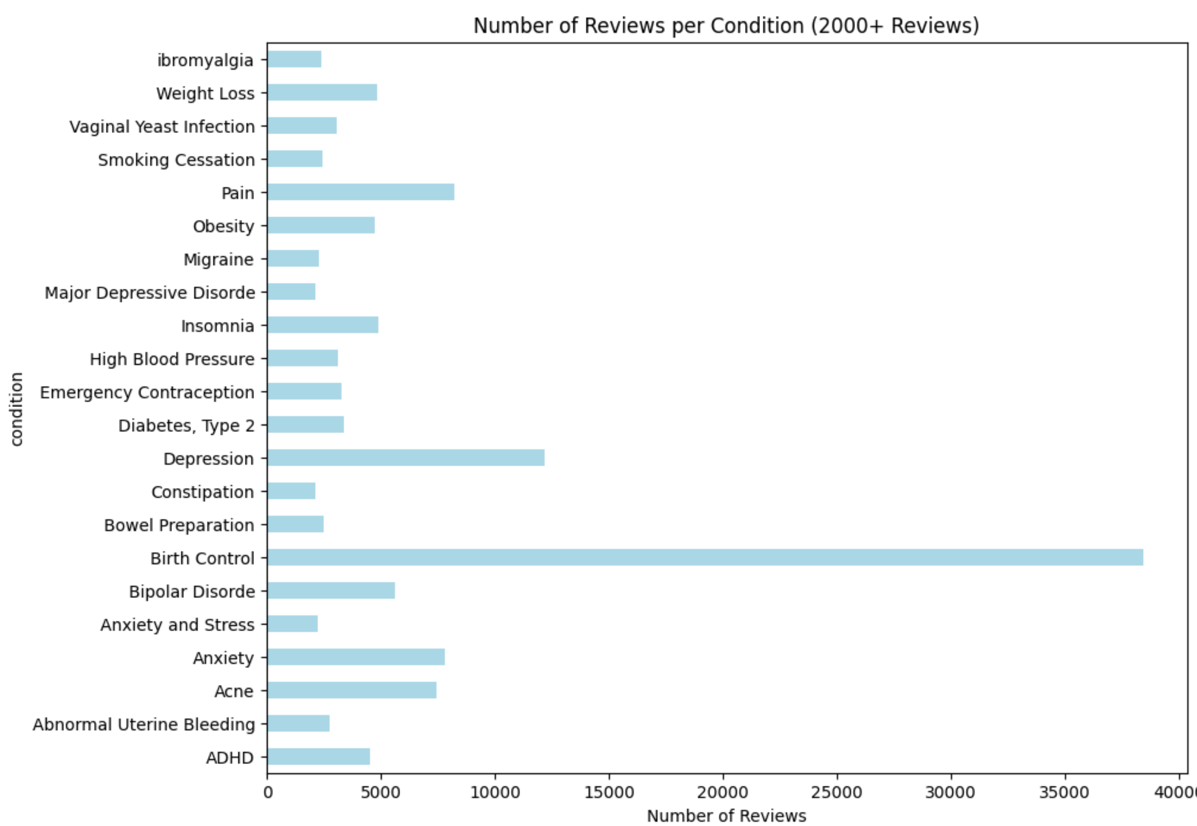


In [9]:
```python
# Aggregating data by month (number of reviews and average rating)
temporal_data = data_original.groupby('year_month').agg(
    num_reviews=('review', 'count'),
    avg_rating=('rating', 'mean')
).reset_index()

# Convert year_month back to datetime for proper plotting
temporal_data['year_month'] = temporal_data['year_month'].dt.to_timestamp()

# Plotting temporal trends
fig, ax1 = plt.subplots(figsize=(12, 6))

# Plot number of reviews over time
ax1.plot(temporal_data['year_month'], temporal_data['num_reviews'], color='b
ax1.set_xlabel('Date', fontsize=12)
ax1.set_ylabel('Number of Reviews', fontsize=12, color='blue')
ax1.tick_params(axis='y', labelcolor='blue')

# Create a second y-axis for the average rating
ax2 = ax1.twinx()
ax2.plot(temporal_data['year_month'], temporal_data['avg_rating'], color='gr
ax2.set_ylabel('Average Rating', fontsize=12, color='green')
ax2.tick_params(axis='y', labelcolor='green')

plt.title('Temporal Trend of Reviews and Average Rating', fontsize=15)
plt.grid(True)
```

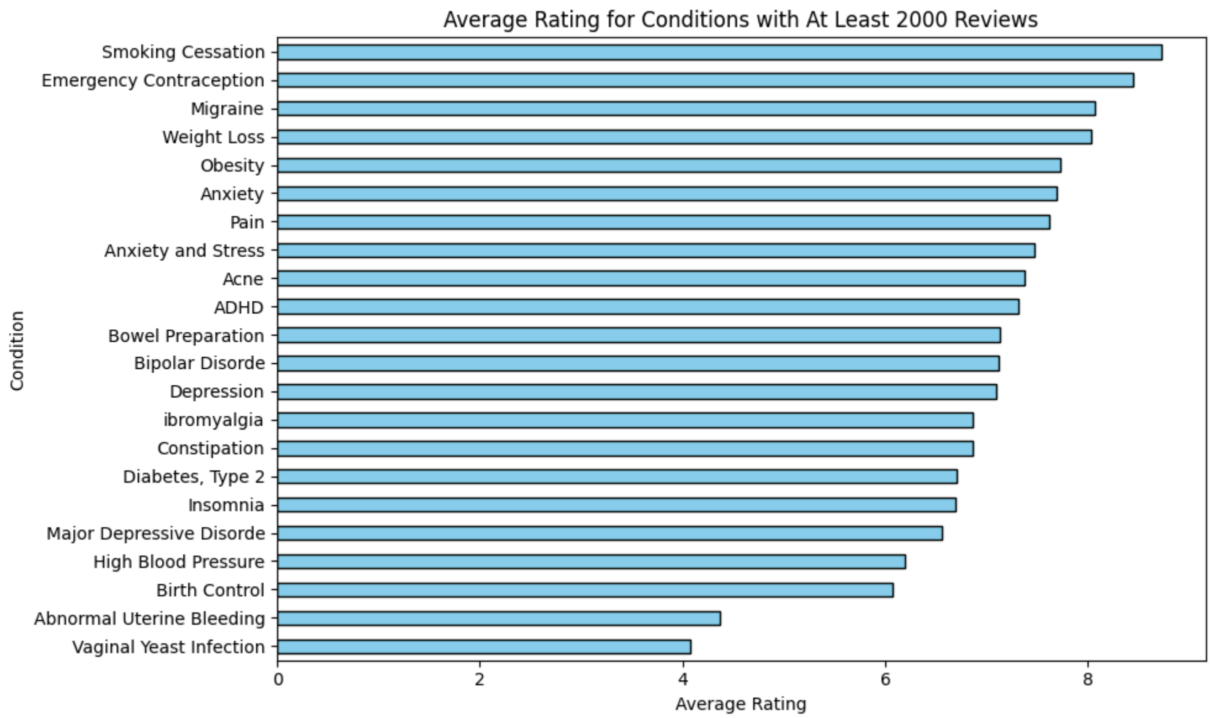
```
plt.tight_layout()

plt.show()
```



In [10]:
```
conditions_large_reviews = condition_review_counts[condition_review_counts >

plt.figure(figsize=(12, 5))
conditions_large_reviews.plot(kind="barh", figsize=(10, 8), color="lightblue
plt.title("Number of Reviews per Condition (2000+ Reviews)")
plt.xlabel("Number of Reviews")
plt.show()
```

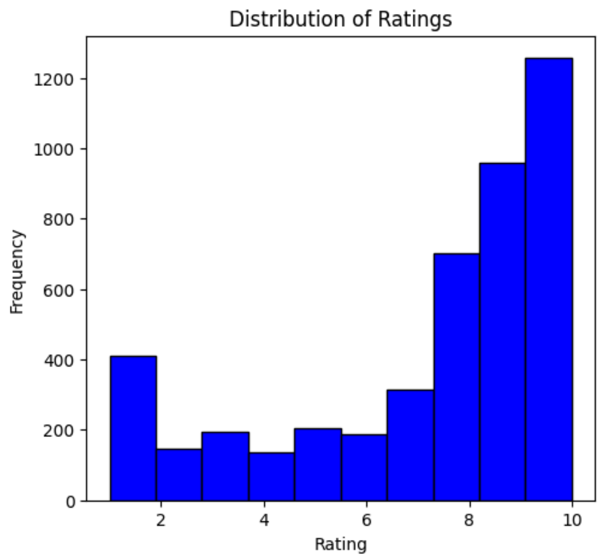


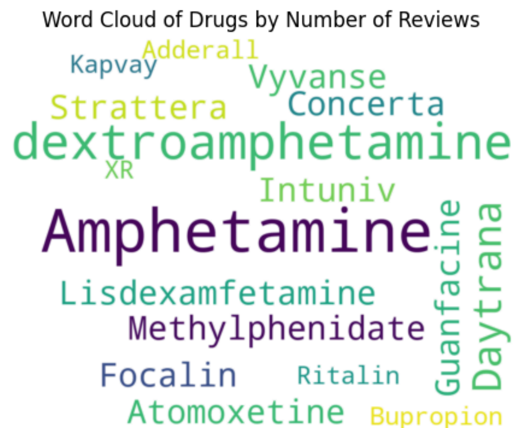
In [11]:
```python
# Filter conditions with at least 2000 reviews
conditions_at_least_2000_reviews = condition_review_counts[condition_review_
filtered_avg_ratings_2000 = avg_rating_per_condition.loc[conditions_at_least

plt.figure(figsize=(10, 6))
filtered_avg_ratings_2000.sort_values().plot(kind="barh", color="skyblue", e
plt.title("Average Rating for Conditions with At Least 2000 Reviews")
plt.xlabel("Average Rating")
plt.ylabel("Condition")
plt.tight_layout()

# Show the plot
plt.show()
```



In [12]:
```python
adhd = data_original[data_original["condition"] == "ADHD"]

plt.figure(figsize=(12, 5))

# Histogram for "rating"
plt.subplot(1, 2, 1)
plt.hist(adhd["rating"], bins=10, color="blue", edgecolor="black")
plt.title("Distribution of Ratings")
plt.xlabel("Rating")
plt.ylabel("Frequency")


# Generate a string of all drug names repeated based on their counts
drug_counts = adhd["drugName"].head(30).value_counts()
drug_string = " ".join([f"{drug} " * count for drug, count in drug_counts.it

# Create the word cloud
wordcloud = WordCloud(width=1000, height=800, background_color="white").gene
```

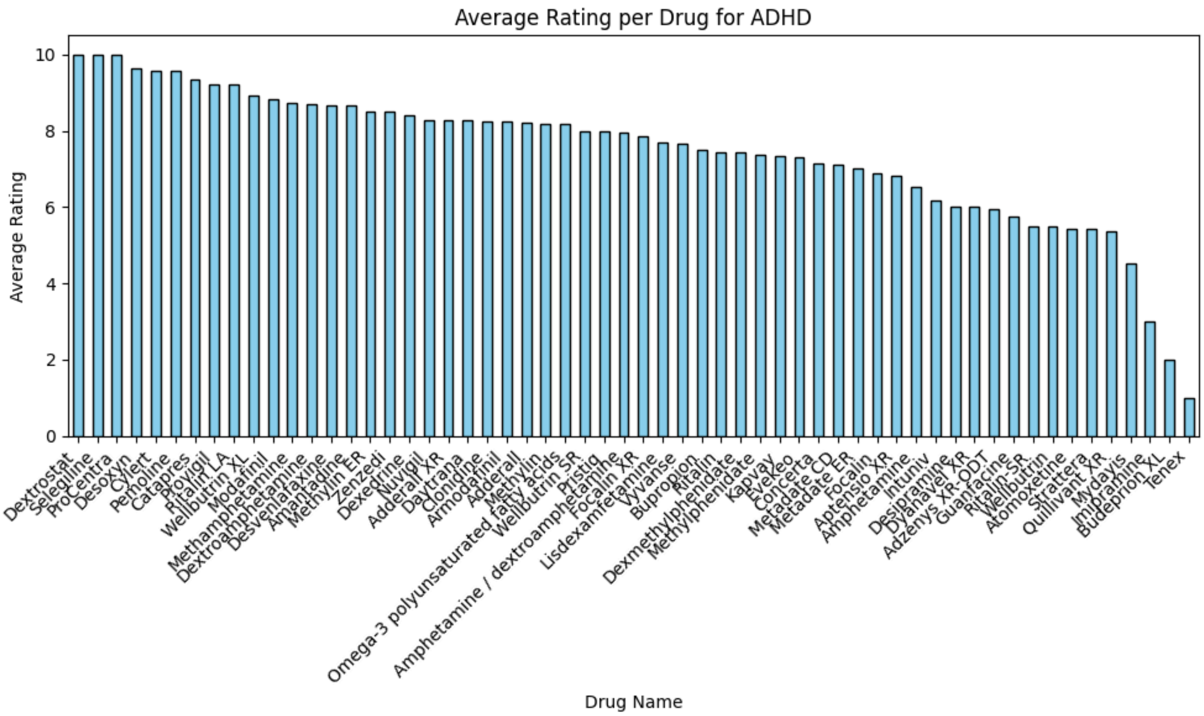
```python
# Plot the word cloud
plt.subplot(1, 2, 2)
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")  # Turn off the axis
plt.title("Word Cloud of Drugs by Number of Reviews")
plt.show()

# Display the plots
plt.tight_layout()
plt.show()
```





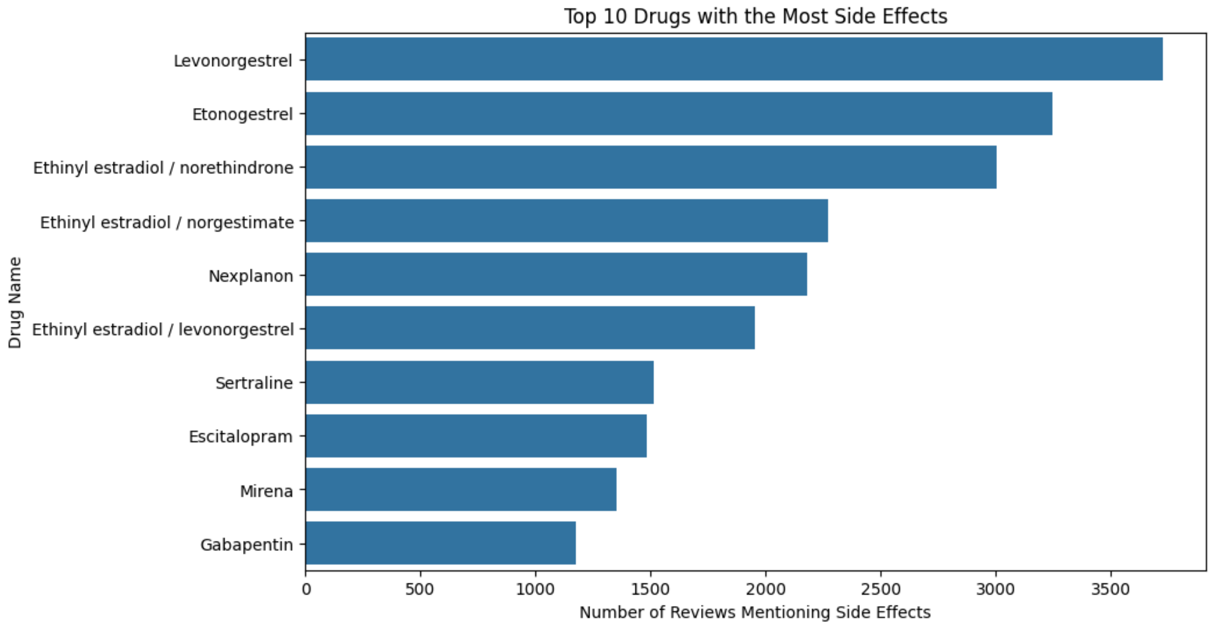

```
<Figure size 640x480 with 0 Axes>
```

In [13]:
```python
# Calculate the average rating for each drug
avg_rating_per_drug = adhd.groupby("drugName")["rating"].mean()

# Plotting the average rating per drug
plt.figure(figsize=(10, 6))
avg_rating_per_drug.sort_values(ascending=False).plot(kind="bar", color="sky
plt.title("Average Rating per Drug for ADHD")
plt.xlabel("Drug Name")
plt.ylabel("Average Rating")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()

# Show the plot
plt.show()
```



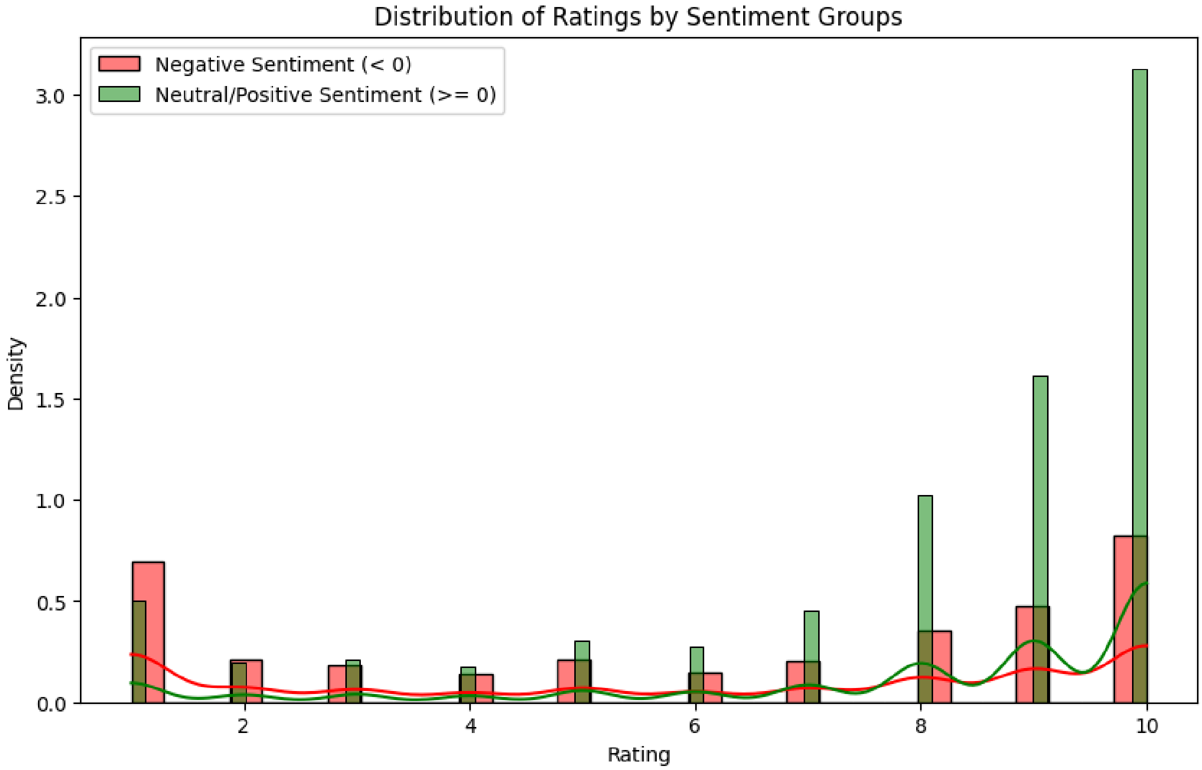
In [14]:

```
# Import local module helper function that checks if side effects are mentio
from utils.side_effects import check_side_effects

# Apply the function to the review text and create the 'side_effects' column
data_original['side_effects'] = data_original['review'].apply(check_side_eff

# Check the distribution of the new target variable
side_effect_counts = data_original['side_effects'].value_counts()
#print(f"Side Effect Target Distribution:\n{side_effect_counts}")

# Group by drug name and sum the 'side_effects' column to count how many rev
side_effects_by_drug = data_original.groupby('drugName')['side_effects'].sum

# Sort the results to find the drugs with the most side effects
side_effects_by_drug_sorted = side_effects_by_drug.sort_values(by='side_effe

# Plot the top 10 drugs with the most side effects
plt.figure(figsize=(10, 6))
sns.barplot(x='side_effects', y='drugName', data=side_effects_by_drug_sorted
plt.title('Top 10 Drugs with the Most Side Effects')
plt.xlabel('Number of Reviews Mentioning Side Effects')
plt.ylabel('Drug Name')
plt.show()
```



In [15]:
```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from sklearn.model_selection import train_test_split
sid = SentimentIntensityAnalyzer()

# Split the data into 75% training and 25% testing
train_df, test_df = train_test_split(data_original, test_size=0.25, random_s
# Preprocess text for sentiment analysis (get sentiment score from review te
data_original['sentiment'] = data_original['review'].apply(lambda x: sid.pol

# Separate the data into two groups: negative (sentiment < 0) and neutral/po
negative = data_original[data_original['sentiment'] < 0]
neutral_positive = data_original[data_original['sentiment'] >= 0]

avg_rating_negative = negative['rating'].mean()
avg_rating_neutral_positive = neutral_positive['rating'].mean()

# Print the results
print(f"Average rating for negative sentiment (sentiment < 0): {avg_rating_r
print(f"Average rating for neutral/positive sentiment (sentiment >= 0): {avg

# plot the distribution of ratings for each sentiment group
plt.figure(figsize=(10, 6))
sns.histplot(negative['rating'], color='red', label='Negative Sentiment (< 0
sns.histplot(neutral_positive['rating'], color='green', label='Neutral/Posit
plt.title('Distribution of Ratings by Sentiment Groups')
plt.xlabel('Rating')
plt.ylabel('Density')
plt.legend()
plt.show()
```

```
Average rating for negative sentiment (sentiment < 0): 6.08
Average rating for neutral/positive sentiment (sentiment >= 0): 7.93
```

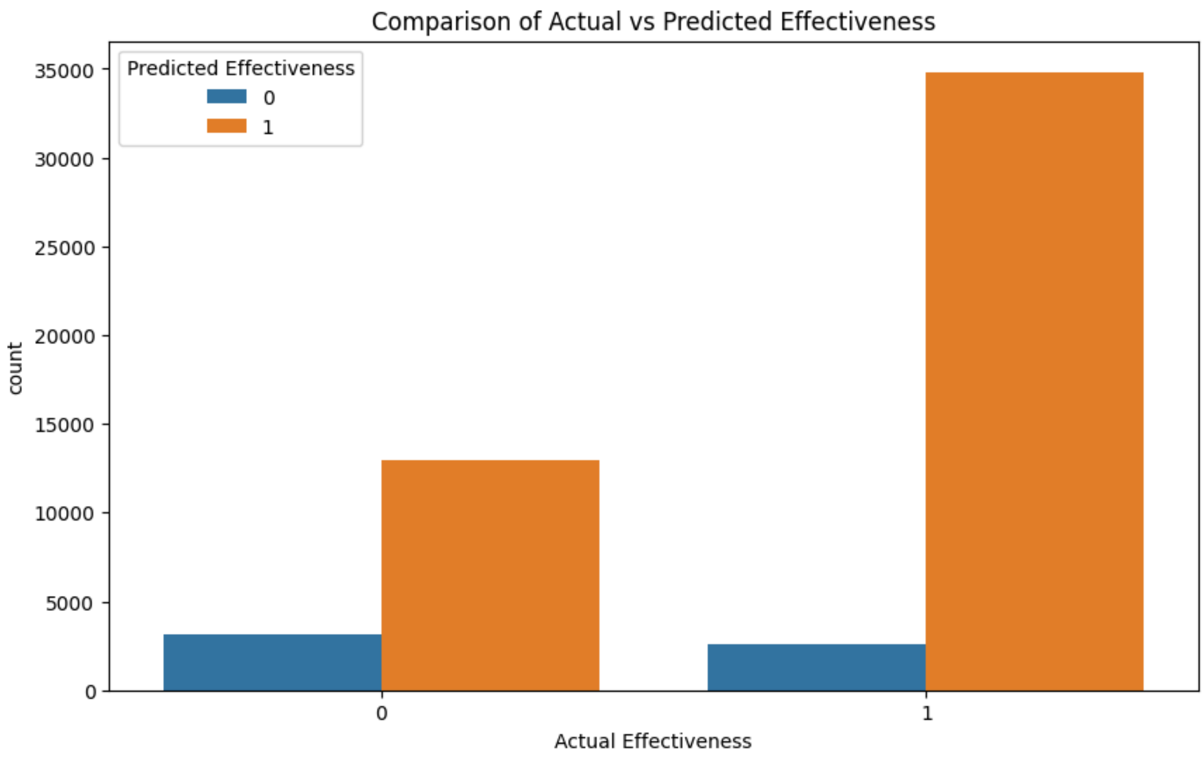


In [16]:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Define effectiveness for training data, here we assumed the ratings > 5 me
# which can be adjusted withe the previous section of sentiment + rating
train_df['effective'] = train_df.apply(
    lambda row: 1 if row['rating'] > 5 else 0, axis=1
)

# Define effectiveness for test data in the same way
test_df['effective'] = test_df.apply(
    lambda row: 1 if row['rating'] > 5 else 0, axis=1
)

# Handle missing data
train_df.dropna(subset=['drugName', 'condition', 'rating'], inplace=True)
test_df.dropna(subset=['drugName', 'condition', 'rating'], inplace=True)

# Encode categorical variables (Drug name, Condition)
le_drug = LabelEncoder()
le_condition = LabelEncoder()

train_df['drug_encoded'] = le_drug.fit_transform(train_df['drugName'])
train_df['condition_encoded'] = le_condition.fit_transform(train_df['conditi

# Define a safe transformation function to handle unseen labels
def safe_transform(encoder, data, default_value=-1):
    return np.array([default_value if label not in encoder.classes_ else enc

# Apply safe transformation on the test data to handle unseen drugs and conc
test_df['drug_encoded'] = safe_transform(le_drug, test_df['drugName'])
```

```python
test_df['condition_encoded'] = safe_transform(le_condition, test_df['conditi

# Define the input features (drug name and condition) and the target variabl
X_train = train_df[['drug_encoded', 'condition_encoded']]
y_train = train_df['effective']

X_test = test_df[['drug_encoded', 'condition_encoded']]
y_test = test_df['effective']

# Train a Random Forest classifier using drug name and condition as input fe
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Make predictions on the test set using drug name and condition
y_pred = clf.predict(X_test)

# Evaluate the model
print(f"Train Accuracy: {accuracy_score(y_train, clf.predict(X_train))}")
print(f"Test Accuracy: {accuracy_score(y_test, y_pred)}")
print(classification_report(y_test, y_pred))

# Create a comparison table of actual vs predicted values
comparison_df = pd.DataFrame({
    'Drug Name': test_df['drugName'],
    'Condition': test_df['condition'],
    'Actual Effectiveness': y_test,
    'Predicted Effectiveness': y_pred
})

print(comparison_df.head())

# Plot the predicted effectiveness for a few drug-condition pairs
plt.figure(figsize=(10, 6))
sns.countplot(x='Actual Effectiveness', hue='Predicted Effectiveness', data=
plt.title('Comparison of Actual vs Predicted Effectiveness')
plt.show()
```

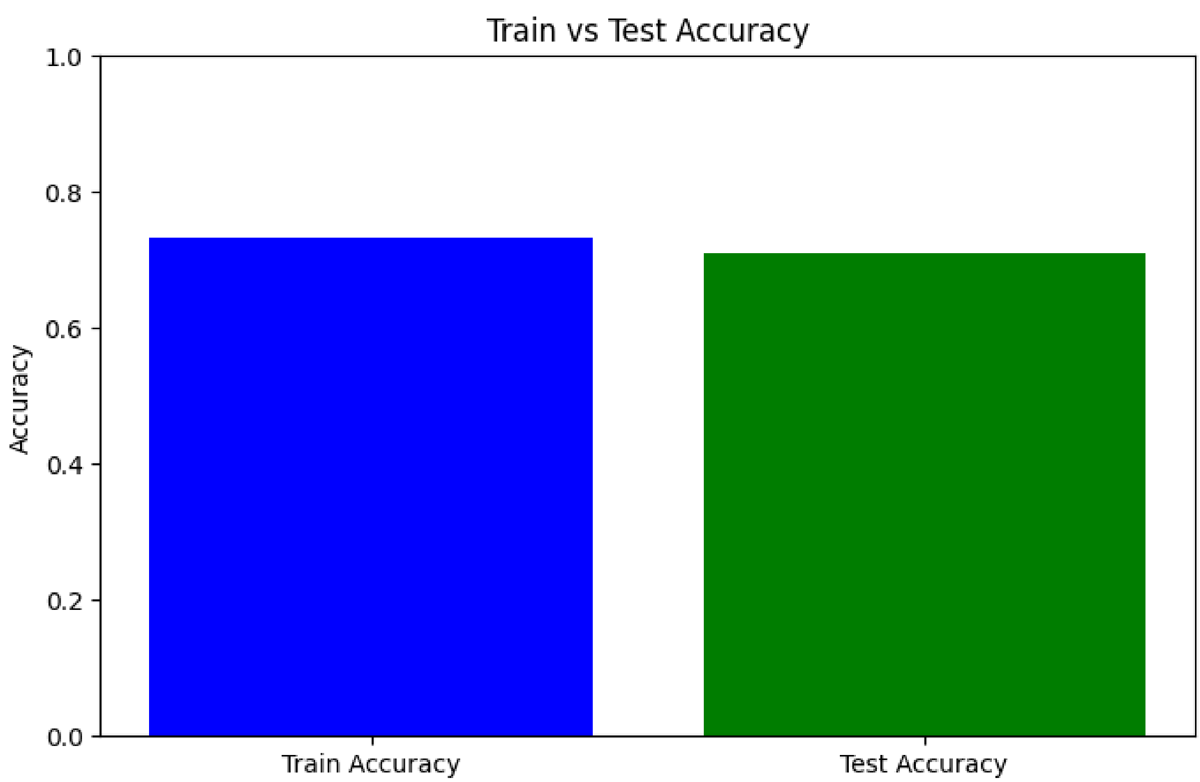
```
Train Accuracy: 0.7325016676953385
Test Accuracy: 0.7094710855090895
              precision    recall  f1-score   support

           0       0.54      0.19      0.29     16035
           1       0.73      0.93      0.82     37433

    accuracy                           0.71     53468
   macro avg       0.64      0.56      0.55     53468
weighted avg       0.67      0.71      0.66     53468

            Drug Name           Condition  Actual Effectiveness  \
148619         Epiduo                Acne                     1
90835     Levofloxacin         Bronchitis                     0
194827      Amlodipine  High Blood Pressure                    1
113551         Dapsone                Acne                     0
89748    Amitriptyline           Insomnia                     1

        Predicted Effectiveness
148619                        1
90835                         0
194827                        0
113551                        1
89748                         1
```

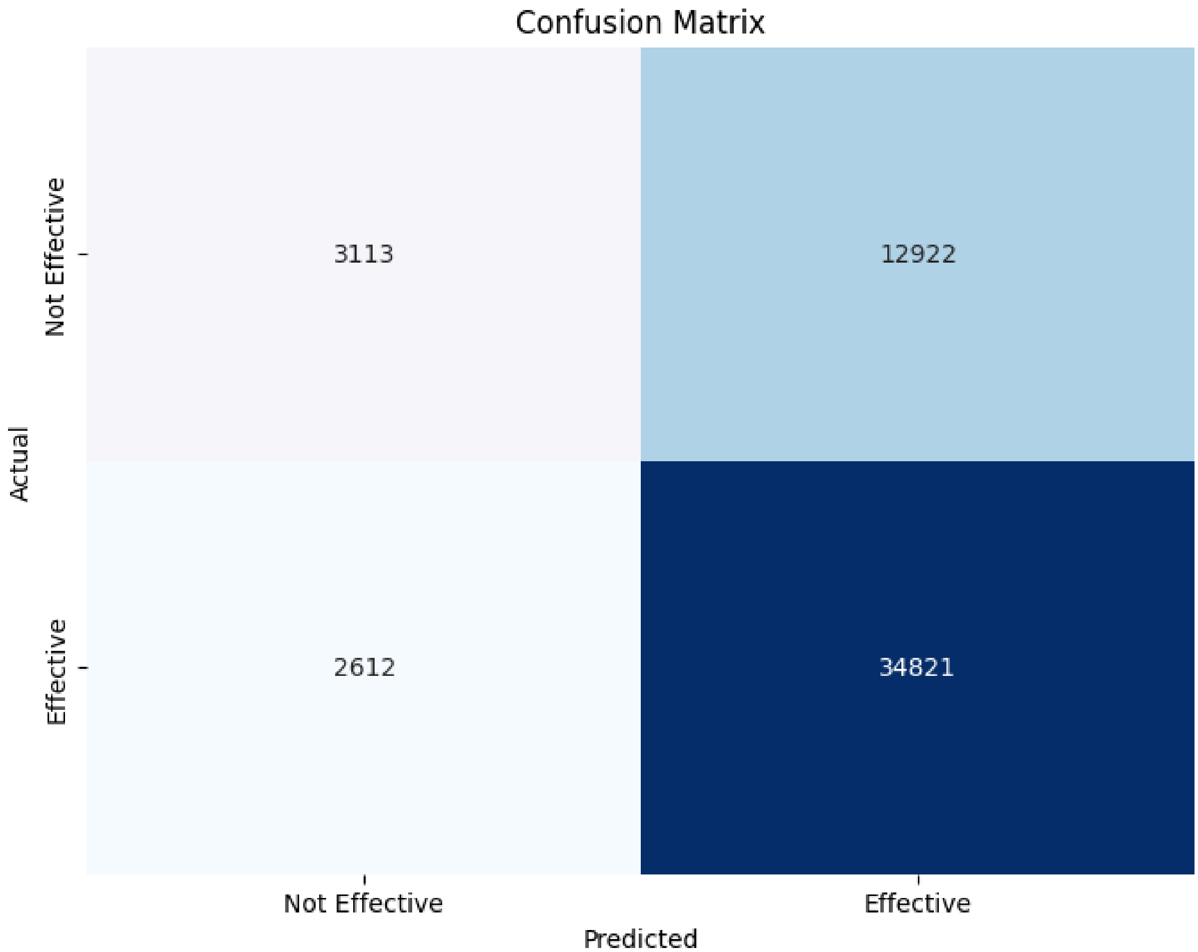


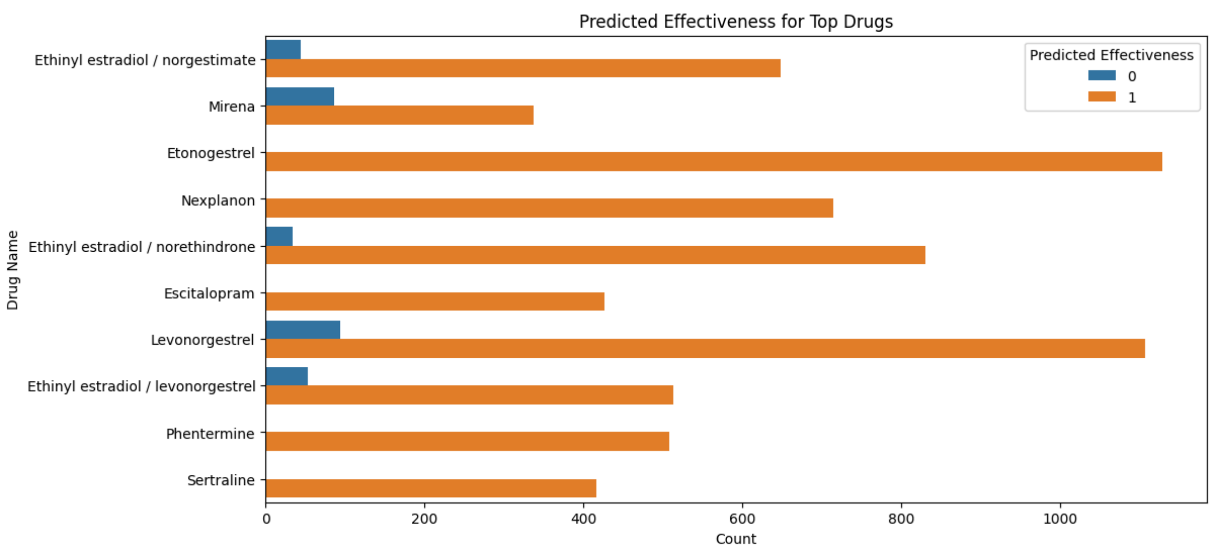
In [17]:

```
#Some visualizations, we can pick and chose what we want to keep
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Train and Test Accuracy as a bar plot
plt.figure(figsize=(8, 5))
accuracy_values = [accuracy_score(y_train, clf.predict(X_train)), accuracy_s
plt.bar(['Train Accuracy', 'Test Accuracy'], accuracy_values, color=['blue',
```

```
plt.title('Train vs Test Accuracy')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.show()

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=[
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Bar plot of Predicted Effectiveness for different drugs
plt.figure(figsize=(12, 6))
top_drugs = comparison_df['Drug Name'].value_counts().index[:10]
sns.countplot(y='Drug Name', hue='Predicted Effectiveness', data=comparison_
plt.title('Predicted Effectiveness for Top Drugs')
plt.xlabel('Count')
plt.ylabel('Drug Name')
plt.legend(title='Predicted Effectiveness', loc='upper right')
plt.show()
```





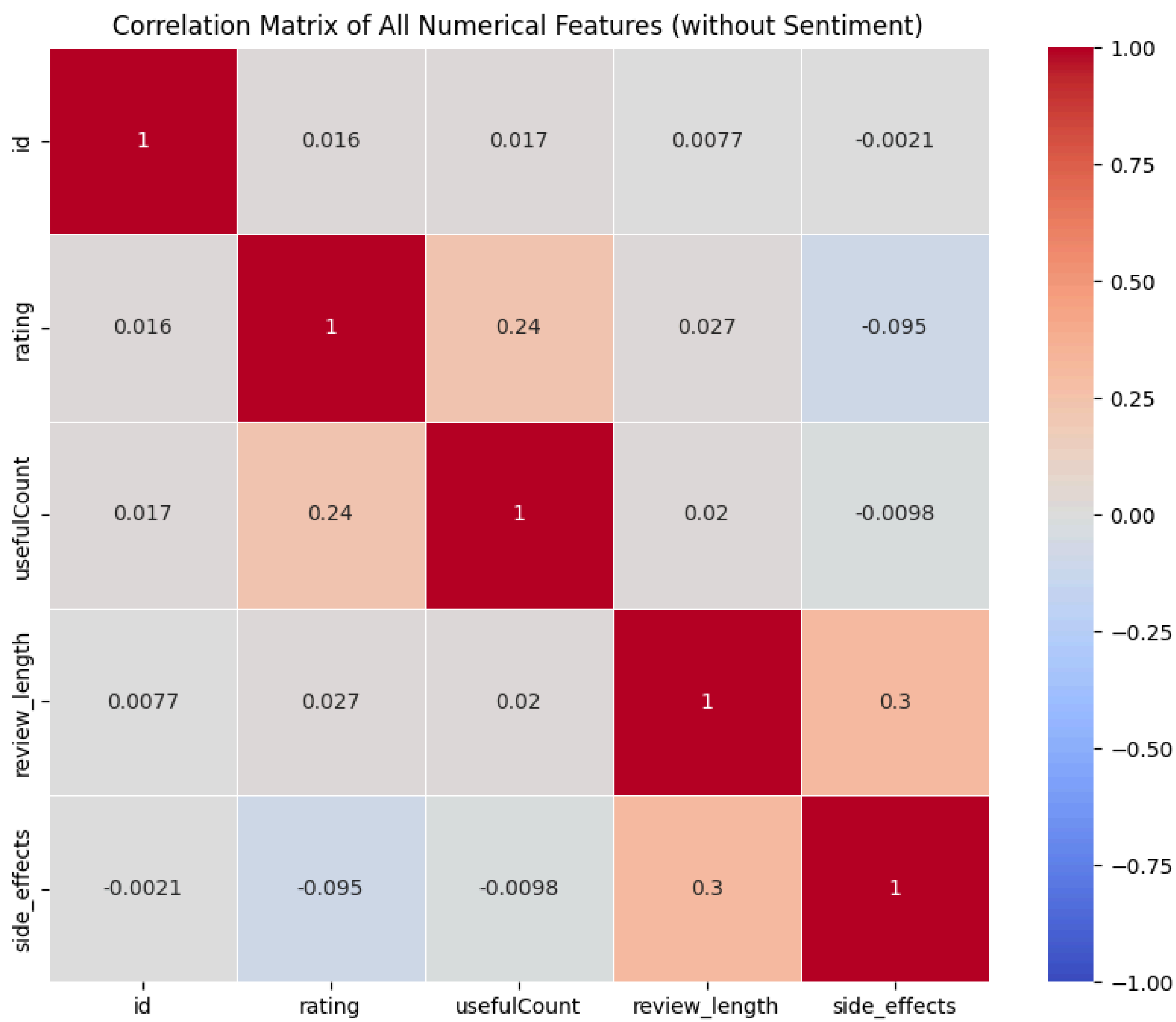


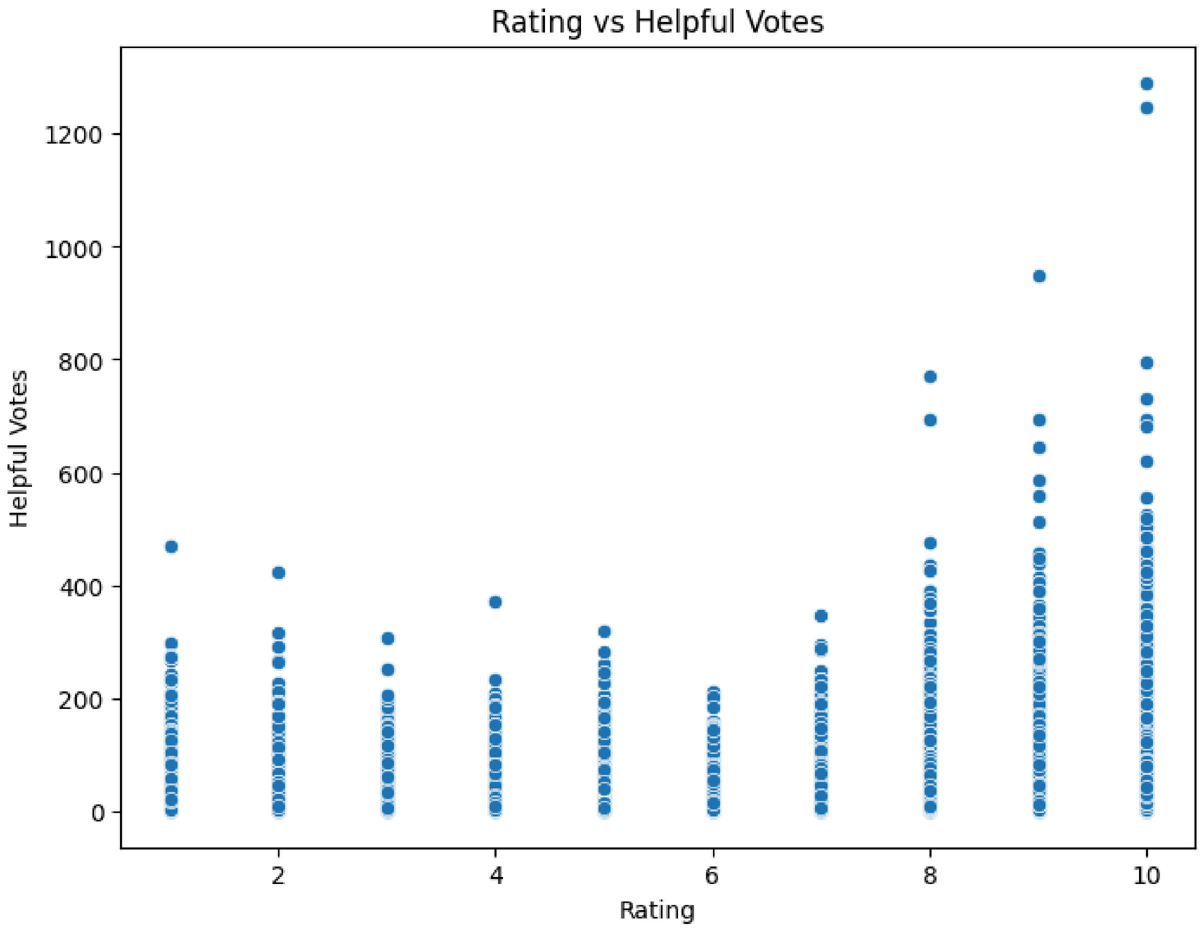
In [18]:

```
# Some additional descriptive analysis that could be useful:
# Select all numerical columns except 'sentiment'
numerical_columns = data_original.select_dtypes(include=['float64', 'int64']
numerical_columns = numerical_columns.drop('sentiment', errors='ignore')  #

# Calculate the correlation matrix for the selected numerical features
correlation_matrix = data_original[numerical_columns].corr()

# Display the correlation matrix
print("Correlation Matrix (without Sentiment):")
```

```python
print(correlation_matrix)

# Plot a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1
plt.title('Correlation Matrix of All Numerical Features (without Sentiment)'
plt.show()

# Scatter plot for rating vs usefulCount (helpful votes)
plt.figure(figsize=(8, 6))
sns.scatterplot(x='rating', y='usefulCount', data=data_original)
plt.title('Rating vs Helpful Votes')
plt.xlabel('Rating')
plt.ylabel('Helpful Votes')
plt.show()
```

```
Correlation Matrix (without Sentiment):
                     id    rating  usefulCount  review_length  side_effects
id             1.000000  0.015925     0.017024       0.007668     -0.002079
rating         0.015925  1.000000     0.235121       0.026788     -0.095272
usefulCount    0.017024  0.235121     1.000000       0.019912     -0.009833
review_length  0.007668  0.026788     0.019912       1.000000      0.303126
side_effects  -0.002079 -0.095272    -0.009833       0.303126      1.000000
```



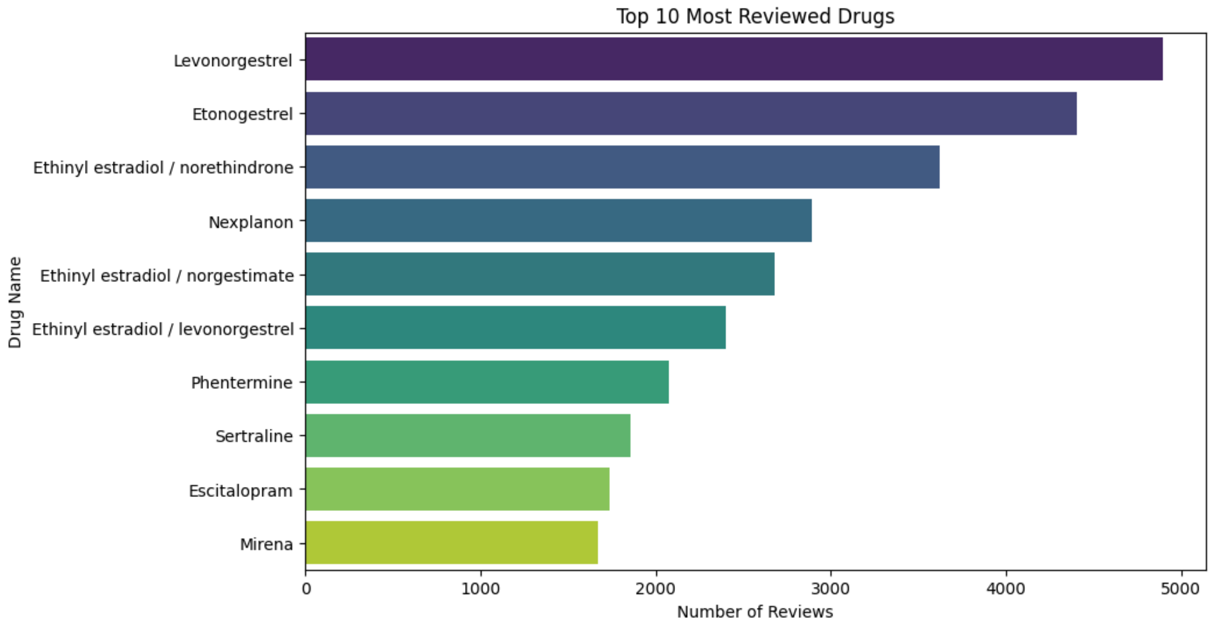


In [19]:

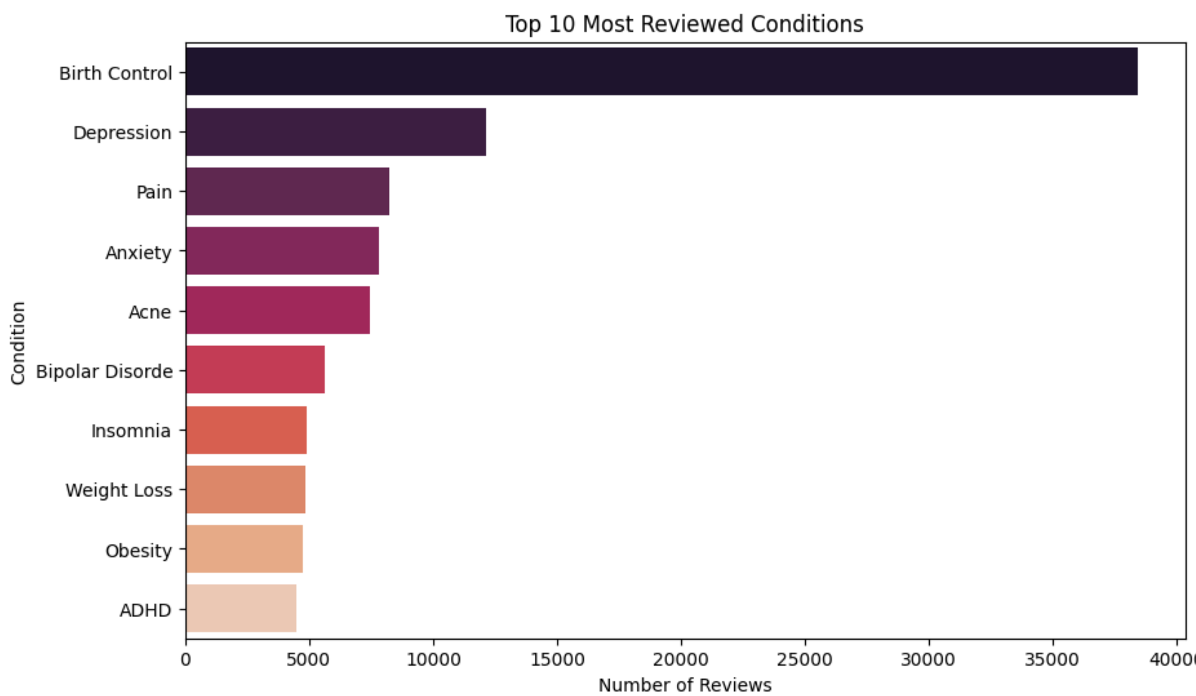
```
# Top 10 most common drugs
top_drugs = data_original['drugName'].value_counts().head(10)
print("Top 10 Most Reviewed Drugs:")
print(top_drugs)

# Visualize the top 10 most common drugs
plt.figure(figsize=(10, 6))
sns.barplot(x=top_drugs.values, y=top_drugs.index, hue=top_drugs.index, lege
plt.title('Top 10 Most Reviewed Drugs')
plt.xlabel('Number of Reviews')
plt.ylabel('Drug Name')
plt.show()

# Top 10 most common conditions
top_conditions = data_original['condition'].value_counts().head(10)
print("Top 10 Most Reviewed Conditions:")
print(top_conditions)

# Visualize the top 10 conditions
plt.figure(figsize=(10, 6))
sns.barplot(x=top_conditions.values, y=top_conditions.index, hue=top_conditi
plt.title('Top 10 Most Reviewed Conditions')
plt.xlabel('Number of Reviews')
plt.ylabel('Condition')
plt.show()

# Number of reviews per drug and condition
```

```python
reviews_per_drug_condition = data_original.groupby(['drugName', 'condition']
top_drug_condition = reviews_per_drug_condition.sort_values(by='num_reviews'

print("Top 10 Drug-Condition Pairs by Number of Reviews:")
print(top_drug_condition)

# Visualize the top drug-condition pairs
plt.figure(figsize=(10, 6))
sns.barplot(x='num_reviews', y='drugName', hue='condition', data=top_drug_co
plt.title('Top 10 Drug-Condition Pairs by Number of Reviews')
plt.xlabel('Number of Reviews')
plt.ylabel('Drug Name')
plt.legend(title='Condition', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()

# Plot the distribution of sentiment scores
plt.figure(figsize=(10, 6))
sns.histplot(data_original['sentiment'], bins=20, kde=True, color='purple')
plt.title('Sentiment Score Distribution')
plt.xlabel('Sentiment Score')
plt.ylabel('Count')
plt.show()

# Show how many reviews have positive, negative, or neutral sentiment
print("Sentiment Distribution:")
sentiment_labels = ['Negative', 'Neutral', 'Positive']
sentiment_distribution = [
    (data_original['sentiment'] < 0).sum(),
    (data_original['sentiment'] == 0).sum(),
    (data_original['sentiment'] > 0).sum()
]
for label, count in zip(sentiment_labels, sentiment_distribution):
    print(f"{label}: {count} reviews")
```

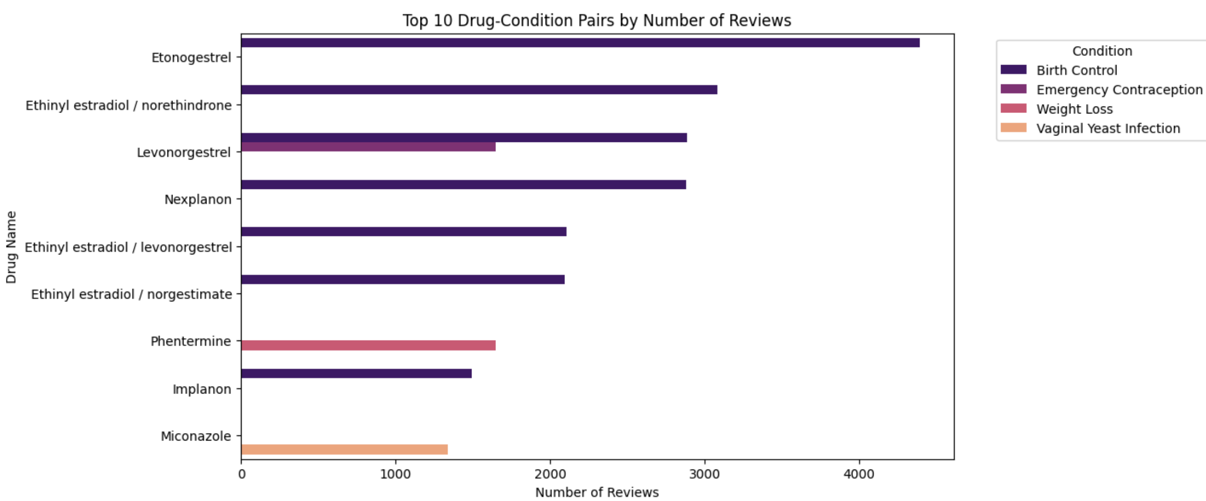
```
Top 10 Most Reviewed Drugs:
drugName
Levonorgestrel                          4896
Etonogestrel                            4402
Ethinyl estradiol / norethindrone       3619
Nexplanon                               2892
Ethinyl estradiol / norgestimate        2682
Ethinyl estradiol / levonorgestrel      2400
Phentermine                             2077
Sertraline                              1859
Escitalopram                            1739
Mirena                                  1673
Name: count, dtype: int64
```

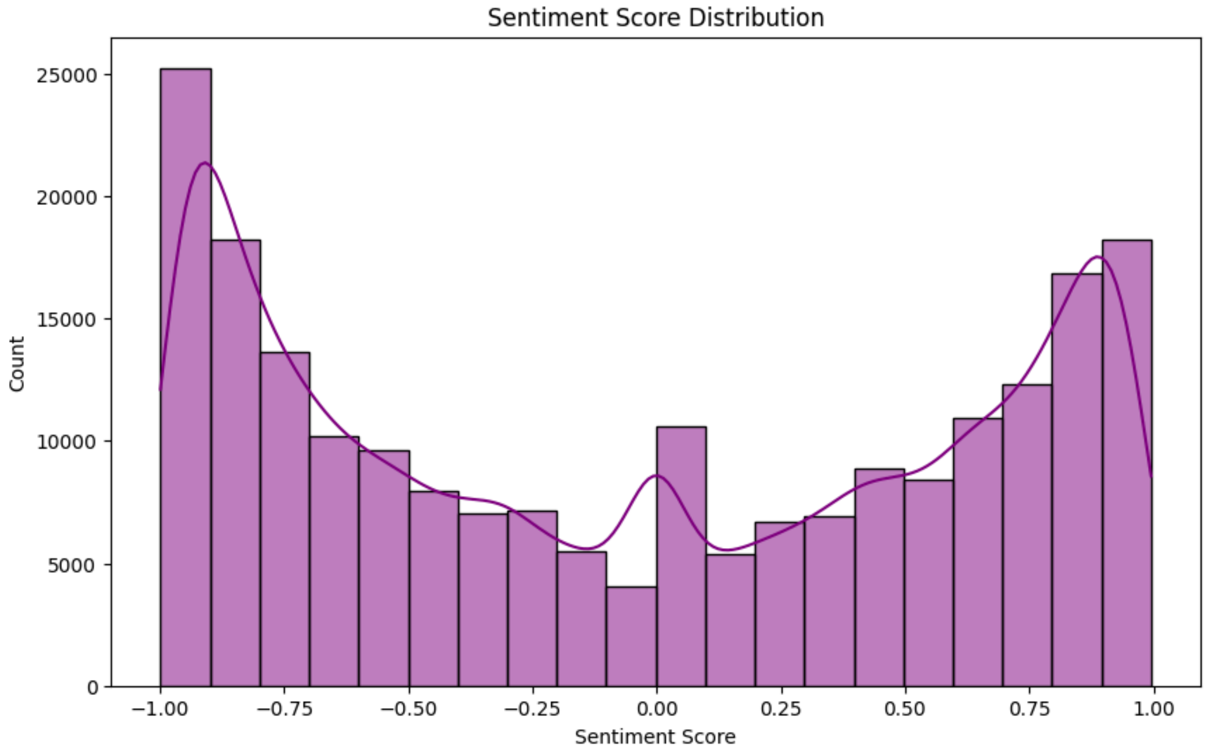



```
Top 10 Most Reviewed Conditions:
condition
Birth Control      38436
Depression         12164
Pain                8245
Anxiety             7812
Acne                7435
Bipolar Disorde     5604
Insomnia            4904
Weight Loss         4857
Obesity             4757
ADHD                4509
Name: count, dtype: int64
```



```
Top 10 Drug-Condition Pairs by Number of Reviews:
                                drugName                condition  num_revie
ws
3307                        Etonogestrel            Birth Control         43
94
3283   Ethinyl estradiol / norethindrone            Birth Control         30
81
4790                      Levonorgestrel            Birth Control         28
84
6012                           Nexplanon            Birth Control         28
83
3274  Ethinyl estradiol / levonorgestrel            Birth Control         21
07
3292    Ethinyl estradiol / norgestimate            Birth Control         20
97
4791                      Levonorgestrel  Emergency Contraception         16
51
6693                         Phentermine              Weight Loss         16
50
4233                            Implanon            Birth Control         14
96
5527                          Miconazole  Vaginal Yeast Infection         13
38
```






```
Sentiment Distribution:
Negative: 108677 reviews
Neutral: 6728 reviews
Positive: 98464 reviews
```

In [20]:
```
import ipywidgets as widgets
from IPython.display import display

# Define effectiveness using percentiles of the ratings to add more variance
# Compute percentiles and scale effectiveness
data_original['percentile_effectiveness'] = data_original.groupby('conditior

# Group by condition and drug, then calculate the mean percentile effectiver
effectiveness_by_condition = data_original.groupby(['condition', 'drugName']
    average_rating=('rating', 'mean'),
    confidence=('percentile_effectiveness', 'mean')  # Now using percentile
).reset_index()

# Function to get the best drug for a given condition
def get_best_drug(condition):
    condition_data = effectiveness_by_condition[effectiveness_by_condition['

    if condition_data.empty:
        return "No data available", 0

    best_drug = condition_data.sort_values(by='confidence', ascending=False)

    return best_drug['drugName'], best_drug['confidence']  # Return the conf

# Create a dropdown widget for selecting a condition
condition_dropdown = widgets.Dropdown(
    options=data_original['condition'].unique(),
    description='Condition:',
```

```python
    disabled=False
)

# Create a label widget to display the best drug and confidence
result_label = widgets.Label(value="Select a condition to see the best drug"

# Function to update the result when a condition is selected
def on_condition_change(change):
    if change['type'] == 'change' and change['name'] == 'value':
        selected_condition = change['new']
        best_drug, confidence = get_best_drug(selected_condition)
        result_label.value = f"Best Drug: {best_drug}\nConfidence: {confider

# Attach the function to the dropdown widget
condition_dropdown.observe(on_condition_change)

# Display the dropdown and result label in the notebook
display(condition_dropdown)
display(result_label)
```

```
Dropdown(description='Condition:', options=('Left Ventricular Dysfunction',
'ADHD', 'Birth Control', 'Opiate D…
Label(value='Select a condition to see the best drug')
```